Communication Avoiding ILU0 Preconditioner

Laura Grigori, Sophie Moufawad

Communication Avoiding ILU0 Preconditioner rapport de recherche Inria

Laura Grigori, Sophie Moufawad

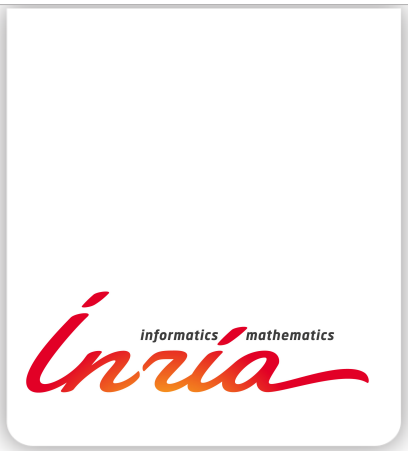
RESEARCH REPORT

N° 8266

March 2013

Project-Teams
ALPINES

ISSN 0249-6399 ISRN INRIA/RR--8266--FR+ENG



Communication Avoiding ILU0 Preconditioner
rapport de recherche
Inria

Laura Grigori*, Sophie Moufawad *

Équipes-Projets
ALPINES

Rapport de recherche n° 8266 — March 2013 — 18 pages

Résumé : Dans cet article, nous présentons "Communication-Avoiding ILU0", un préconditionneur pour la résolution des systèmes linéaires d'équations de très grande taille. Ce préconditionneur permet de minimiser les communications, il est possible d'éffectuer s itérations d'une méthode itérative sans communication.

Mots-clés : algèbre linéaire, méthodes itératives

* INRIA

RESEARCH CENTRE
PARIS – ROCQUENCOURT

Domaine de Voluceau, - Rocquencourt
B.P. 105 - 78153 Le Chesnay Cedex

Communication Avoiding ILU0 Preconditioner

Abstract: In this paper we present a communication avoiding ILU0 preconditioner for solving large linear systems of equations by using iterative Krylov subspace methods. Recent research has focused on communication avoiding Krylov subspace methods based on so called s-step methods. However there is no communication avoiding preconditioner yet, and this represents a serious limitation of these methods. Our preconditioner allows to perform s iterations of the iterative method with no communication, through ghosting some of the input data and performing redundant computation. It thus reduces data movement by a factor s between different levels of the memory hierarchy in a serial computation and between different processors in a parallel computation. To avoid communication, an alternating reordering algorithm is introduced for structured matrices, that requires the input matrix to be ordered by using nested dissection. We show that the reordering does not affect the convergence rate of the ILU0 preconditioned system as compared to nested dissection ordering, while it reduces data movement and should improve the expected time needed for convergence.

Key-words: minimizing communication, linear algebra, iterative methods

1 Introduction

Many scientific problems require the solution of systems of linear equations of the form $Ax = b$, where the input matrix A is very large and sparse. We focus in this paper on solving such systems using Krylov subspace methods, as GMRES [13], CG [8], in a communication avoiding way. These methods iterate from a starting vector y until convergence or stagnation, by using projections onto the Krylov subspace $K(A, y) = span[y, Ay, A^2y, ...]$. In the parallel case, the input matrix is distributed over processors, and each iteration involves multiplying the input matrix with a vector, followed by an orthogonalizion process. Both these operations require communication among processors. Since A is usually very sparse, the communication dominates the overall cost of the iterative methods when the number of processors is increased to a large number. More generally, on current machines the cost of moving data is much higher than the cost of arithmetic operations, and this gap is expected to continue to increase exponentially. As a result, communication is often a bottleneck in numerical algorithms.

In a quest to address the communication problem, recent research has focused on reformulating linear algebra operations such that the movement of data is significantly reduced, or even minimized as in the case of dense matrix factorizations [3, 7]. Such algorithms are referred to as communication avoiding. The communication avoiding Krylov subspace methods are based on s-step methods [11], which reformulate the iterative method such that the dependencies between certain iterations are eliminated. In short, the reformulation allows, by unrolling s iterations of the Krylov subspace method and ghosting some of the data, to compute s vectors of the basis without communication, followed by an orthogonalization step. The orthogonalization step is performed by using TSQR, a communication optimal QR factorization algorithm [3]. By performing s steps at once, the number of messages exchanged between different processors is reduced by a factor s. In the case of a sequential algorithm, both the number of messages and the volume of communication are reduced by a factor s. The communication avoiding version of GMRES (CA-GMRES), introduced in [11, 9], can be seen as a generalization of s-step GMRES [14, 6]. It mainly uses two communication avoiding kernels: the matrix power kernel [4] and TSQR factorization [3]. The derivation and implementation of CA-CG can be found in [11, 9], while other Krylov subspace methods, as CA-CGS, CA-BICG, and CA-BICGStab methods, were introduced recently in [2].

However, except a discussion in [9], there is no available preconditioner that can be used with s-step methods. This is a serious limitation of these methods, since for difficult problems, Krylov subspace methods without preconditioner can be very slow or even might not converge. Our goal is to design communication avoiding preconditioners that should be efficient in accelerating the iterative method and should also minimize communication. In other words, given a preconditioner M, the preconditioned system with its communication avoiding version $M_{ca}^{-1}Ax = M_{ca}^{-1}b$ should have the same order of convergence as the original preconditioned system $M^{-1}Ax = M^{-1}b$ and reduce communication. This is a challenging problem, since applying a preconditioner on its own may, and in general will, require extra communication. Since the construction of M represents typically an important part of the overall runtime of the linear solver, we focus on both minimizing communication during the construction of M and during its application to a vector at each iteration of the linear solver. The incomplete LU factorization (ILU) is a black-box widely used preconditioner, that can be used on its own or as a building block of other preconditioners as domain decomposition methods. The ILU preconditioner is written as $M = LU$, where L and U are the incomplete factors of the input matrix A. This preconditioner is obtained by computing a direct LU factorization of the matrix A, and by dropping some of the elements during the decomposition, based on either their numerical value or their relation with respect to the graph of the input matrix A.

In this paper we introduce CA-ILU0, a communication avoiding ILU0 preconditioner. We will first start by adapting the matrix power kernel to the ILU preconditioned system to obtain the ILU matrix power kernel in section 2. Each vector of this kernel is obtained by computing $((LU)^{-1}Ax)$, that is in addition to the matrix-vector multiplication Ax, it uses a forward and a backward substitution. The ILU matrix power kernel, which is designed for any given LU decomposition, does not allow to avoid

communication by itself. That is, if we want to compute s vectors of this kernel with no communication through ghosting some of the data, there are cases when one processor performs an important part of the entire computation. We restrain then our attention to the ILU0 factorization, which has the property that the L and U factors have the same sparsity pattern as the lower triangular part of A and the upper triagular part of A respectively. To obtain a communication avoiding ILU0 preconditioner, we introduce in section 3 a reordering of the input matrix A, which is reflected in the L and U matrices. This reordering allows to avoid communication for s-steps of the matrix vector multiplication ($(LU)^{-1}Ax$). In other words, s backward and forward solves corresponding to a submatrix of A can be performed, without needing any data from other submatrices. This is possible since the CA-ILU0 $(L)^{-1}$ and $(U)^{-1}$ are sparse unlike those of ILU0 (Appendix C). In this paper we will portray our CA-ILU0 preconditioner and its performace using GMRES, but it can be used with other Krylov subspace methods as well. We focus on structured matrices arising from the discretization of partial differential equations on regular grids, and we note that the methods can be extended to unstructured matrices. The CA-ILU0(s) reordering can be used to avoid communication not only in parallel computations (between processors or shared-memory cores or CPU and GPU) but also in sequential computations (between different levels of memory hierarchy). Thus in this paper we will use the term *processor* to indicate the component performing the computation and *fetch* to indicate the movement of data (read, copy or receive message). In section 4 we show that our reordering does not affect the convergence of ILU0 preconditioned GMRES, and we model the expected performance of our preconditioner based on the needed memory and the redundant flops introduced to reduce the communication.

2 ILU Matrix Power Kernel

The algorithm for solving a left-preconditioned system by using Krylov subspace methods is the same as a non-preconditioned system with the exception of the matrix vector multiplications. For example in GMRES we have to compute $y = Ax$ whereas in the preconditioned version we compute $y = M^{-1}Ax$, where M is the preconditioner (refer to [12]). Similarly, the difference between the CA-GMRES and the left preconditioned CA-GMRES is in the s-step matrix vector multiplication, which is refered to as the matrix power kernel. Appendix A presents a simplified code of left preconditioned CA-GMRES. In short, constructing a communication avoiding preconditioner is equivalent to building an $s - step$ preconditioned matrix power kernel which computes the set of s basis vectors $\{(M^{-1}A)^s y_0, (M^{-1}A)^{s-1} y_0, ..., (M^{-1}A)^2 y_0, M^{-1}Ay_0\}$. Each processor has to compute a part α of each of the s vectors without communicating with other processors. In this section we design such an $s - step$ matrix vector multiplication where we assume $M = LU$, irrespective of which LU decomposition is performed to obtain the L and U factors.

The notations used in the paper are the following. Given an $n \times n$ matrix A, we represent its structure by using the graph $G(A) = (V, E)$, where V is a set of vertices and E is a set of edges. A vertex i is associated with each row/column of the matrix A. An oriented edge (j, m) from vertex j to vertex m is associated with each nonzero element $A(j, m) \neq 0$ (vertex j depends on vertex m). Let B be a subgraph of $G(A)$ ($B \subset G(A)$), then $V(B)$ is the set of vertices of B, where $V(B) \subset V(G(A))$, and $E(B)$ is the set of edges of B, where $E(B) \subset E(G(A))$. Let i and j be two vertices of $G(A)$. We say that j is reachable from i if and only if there exists a path of directed edges from i to j. The length of the path is equal to the number of visited vertices excluding i. Let S be any subset of vertices of $G(A)$. We let $R(G(A), S)$ denote the set of vertices reachable from any vertex in S. We assume that $S \subset R(G(A), S)$. We let $R(G(A), S, m)$ denote the set of vertices reachable by paths of length at most m from any vertex in S. We call the set $R(G(A), S, 1)$ the set of adjacency vertices of S in the graph of A and denote it by $Adj(G(A), S)$ or $Adj_A(S)$. We call the set $Adj(G(A), S) - S$ the open set of adjacency vertices of S in the graph of A and denote it by $opAdj(G(A), S)$ or $opAdj_A(S)$. We use matlab notation for matrices and

vectors. For example, given a vector y of size $n \times 1$ and a set of indices α (which correspond to vertices in the graph of A), then $y(\alpha)$ is a vector formed by the subset of the entries of y whose indices belong to α. For a matrix A, $A(\alpha, :)$ is a submatrix formed by the subset of the rows of A whose indices belong to α. Similarly, $A(:, \alpha)$, is a submatrix formed by the subset of the columns of A whoses indices belong to α.

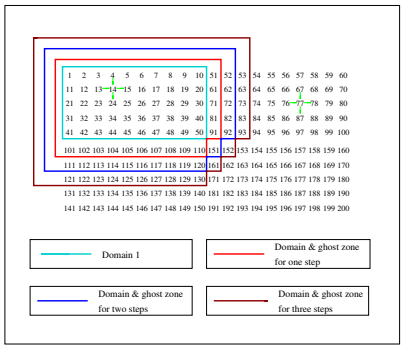
In ILU decomposition with dropping, the structure of the factors L and U can be obtained during the numerical factorization. However, in the case of ILU0 decomposition with no pivoting, the structure of the factor L is the same as the structure of the lower triangular part of A, and the structure of U is the same as the structure of the upper triangular part of A. The graphs of L and U can be simply derived from that of A before the numerical factorization.

S-step matrix power kernel The matrix power kernel can be summarized as follows. First, the data and the work is split between P processors, where each processor is assigned a part α of y_0 ($y_0(\alpha)$) and $A(\alpha, :)$ ($\alpha \subseteq V(G(A))$). Then, each processor has to compute the same part α of $y_1 = Ay_0$, $y_2 = Ay_1$, till $y_s = Ay_{s-1}$ without communicating with other processors. To do so, each processor fetches all the data needed from the neighboring processors, to compute its part α of the s vectors. In general, to compute $y_i(z)$, each processor should have before hand $y_{i-1}(a)$ where $a = Adj_A(z)$. Thus, to compute $y_s(\alpha)$, each processor should have computed $y_{s-1}(a_1)$ where $a_1 = Adj_A(\alpha)$. To compute $y_{s-1}(a_1)$ each processor should have computed $y_{s-2}(a_2)$ where $a_2 = Adj_A(Adj_A(\alpha)) = Adj_A(a_1) = R(G(A), \alpha, 2)$. Similarly, to compute $y_{s-i}(a_i)$ each processor should have $y_{s-i-1}(a_{i+1})$ where $a_{i+1} = Adj_A(a_i) = R(G(A), \alpha, i+1)$. Thus, to compute $y_s(\alpha)$, each processor should fetch the missing data of $y_0(a_s)$ and $A(a_s, :)$ from its neighboring processors, where $a_s = R(G(A), \alpha, s)$. Finally, each processor computes the set $R(G(A), \alpha, s-i)$ of the vector y_i without any communication with the other processors. Since $\alpha \subseteq a_i = R(G(A), \alpha, i)$, it is obvious that the more steps are performed, the more redundant data is ghosted and flops are computed.

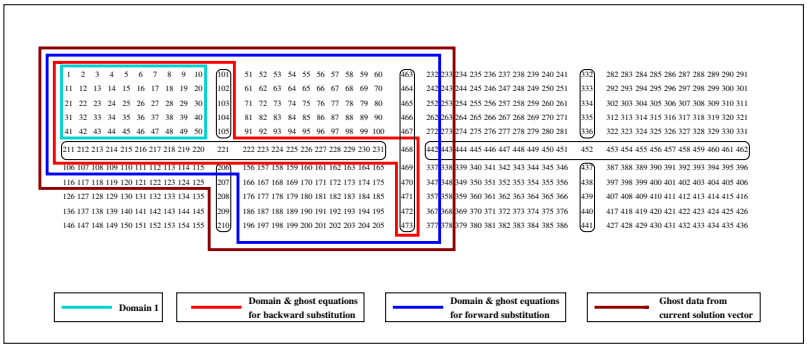
Figure 1a displays the graph of a 2D 5 point-stencil, where the vertices are represented by their indices. For clarity of the figure, the edges are not shown in the graph, but we note that each vertex is connected to its north, south, east and west neighboors. All the following figures of graphs have the same format. The figure shows the needed data for each step on domain 1 where $s = 3$. The vector y_1 is computed on the blue domain using y_0 and A from the brown domain. Similarly, y_2 is computed on the red domain using y_1 and A from the blue domain. Finally, y_3 is computed on Domain 1 using y_2 and A from the red domain. We note that in this structured case, performing one extra step with no communication requires ghosting the neighbors of the currently extended domain. This is reasonable in terms of memory needs and computation.

ILU matrix power kernel Our ILU matrix power kernel is based on the above matrix power kernel with the exception that A is replaced by $(LU)^{-1}A$, since we have a preconditioned system. In practice, $(LU)^{-1}A$ is never computed explicitely, so we don't have any direct information on the graph of $(LU)^{-1}A$ and cannot directly find sets of reachable vertices in this graph. Computing $y_i = (LU)^{-1}Ay_{i-1}$ is equivalent to 3 steps: 1. Compute $f = Ay_{i-1}$
2. Solve $LUy_i = f$ i.e.
   a. Solve $Lz = f$ by forward substitution
   b. Solve $Uy_i = z$ by backward substitution

In the following we describe an algorithm that allows a processor p to perform s steps with no communication, by ghosting parts of A, L, U, and y_0 on processor p before starting the s iterations. To find $y_i(\alpha)$ we have to solve a number of equations of $Uy_i = z$ in addition to equations α. The total set of equations that we need to solve is $\beta = R(G(U), \alpha)$. In other words, we solve the reduced system $U(\beta, \beta)y_i(\beta) = z(\beta)$. We can do so since there are no edges between the vertices β and other vertices by definition of reachable sets. Thus all the column in $U(\beta, :)$ except the β columns are zero columns. To



(a) The graph of a 10-by-20 2D 5 point stencil matrix with s=3, p=4 showing the needed data for each step to compute the matrix power kernel on domain 1, where the overlapping regions outside domain 1 (the ghost zones) indicate redundant computations.



(b) An 11-by-43 grid of a 5-point stencil discretization that is partitioned into 8 subdomains using 7 separators. The data needed to compute one matrix-vector multiplication of the form $y_i = (LU)^{-1}Ay_{i-1}$ on domain1, where the L and U matrices are obtained from ILU0, is shown.

Figure 1

solve the set of equations β of the system $Uy_i = z$, we need to have $z(\beta)$ beforehand. And finding $z(\beta)$ is equivalent to solving the set of equations $\gamma = R(G(L), \beta)$ of $Lz = f$. Similarly, we solve the reduced system $L(\gamma, \gamma)z_i(\gamma) = f(\gamma)$ where $f(\gamma)$ must be available. Computing $f(\gamma)$ is equivalent to computing $A(\gamma, :)x$. However, it must be noted that we do not need to use the whole vector y_i, since for computing this subset of matrix vector multiplication we only need $y_i(\delta)$ where $\delta = Adj(G(A), \gamma)$. Therefore, we compute $A(\gamma, \delta)y_i(\delta)$.

Algorithm 1 Dependencies for s iterations

Input: $G(A)$, $G(L)$, $G(U)$, s: number of steps, α_0: subset of unknowns
Output: Sets β_j, γ_j and δ_j for all $j = 1$ till s

for $j = 1$ *to* s
  Find $\beta_j = R(G(U), \alpha_{j-1})$
  Find $\gamma_j = R(G(L), \beta_j)$
  Find $\delta_j = Adj(G(A), \gamma_j)$
  Set $\alpha_j = \delta_j$
end for

Algorithm 2 ILU Matrix Power Kernel (A, L, U, s, α_0,y_0)

Input: A, L, U, s: number of steps, y_0: input vector, α_0: subset of unknowns
Output: the s vectors $y_k(\alpha_0)$, where $1 \leq k \leq s$

Processor i calls Algorithm 1
Processor i fetches the corresponding parts of A, L, U, y_0
for $j = s$ *to* 1
  Compute $f(\gamma_j) = A(\gamma_j, \delta_j)y_{j-s}(\delta_j)$
  Solve $L(\gamma_j, \gamma_j)z_{j-s+1}(\gamma_j) = f(\gamma_j)$
  Solve $U(\beta_j, \beta_j)y_{j-s+1}(\beta_j) = z(\beta_j)$
  Save $y_{j-s+1}(\alpha_0)$, which is the part that processor p has to compute
end for

To compute one step of $y_i = (LU)^{-1}Ay_{i-1}$, processor p has to fetch $y_{i-1}(\delta)$, $A(\gamma, \delta)$, $L(\gamma, \gamma)$ and $U(\beta, \beta)$. To perform another step, we simply let $\alpha = \delta$ and start again. This procedure is summarized in Algorithm 1. Thus to compute s steps of $y_i(\alpha_0) = [(LU)^{-1}Ay_{i-1}](\alpha_0)$, processor p has to fetch $y0(\delta_s)$, $A(\gamma_s, \delta_s)$, $L(\gamma_s, \gamma_s)$, and $U(\beta_s, \beta_s)$. Note that $\alpha_{i-1} \subseteq \beta_i \subseteq \gamma_i \subseteq \delta_i \subseteq \alpha_i$, for $i = 1$ till s. After

fetching all the data needed, processor p has to compute its part using Algorithm 2. Thus Algorithm 1 has to output all the subsets β_j, γ_j and δ_j for $1 \leq j \leq s$ which will be used in Algorithm 2.

3 Alternating Reordering for CA-ILU0 Matrix Power Kernel

The ILU matrix power kernel presented in section 2 is general and works for any matrices L and U. However, for a communication avoiding method to be efficient, we must have a balance between the number of redundant flops and the amount of communication which was reduced. This reflects in the run time of the algorithm. In other words, if performing three or four steps of CA-ILU, each processor ends up needing all the data and computing almost entirely the vectors y_i, then either we are not exploiting the parallelism of our problem efficiently or the problem is not fit for communication avoiding techniques. This is indeed the case if we apply Algorithm 2 to the 2D 5 point stencil matrix whose graph is presented in figure 1a. Performing only one step of CA-ILU, the processor 1 (which computes Domain 1 in the figure) ends up computing the entire vector y_i and fetching all the matrices A, L and U. This cancels any possible effect of the parallelisation of the problem, and shows that what works for the matrix power kernel of the form $y_i = Ay_{i-1}$ does not work for the same kernel where the multiplication is $y_i = (LU)^{-1}Ay_{i-1}$.

Thus the first step in developing a communication avoiding preconditioner is to partition our domain wisely such that the connection between the subdomains is minimzed (a minimal surface to volume ratio, refer to [4]). We use for our purpose nested dissection [10], a divide and conquer strategy for partitioning undirected graphs. At each step of dissection, a set of vertices that forms a separator is sought, that splits the graph into two disjoint subgraphs once the vertices of the separator are removed. We refer to the two subgraphs as $\Omega_{1,1}$ and $\Omega_{1,2}$, and to the separator as $\Sigma_{1,1}$. The vertices of the first subgraph are numbered first, then those of the second subgraph, and finally those of the separator. The corresponding matrix has the following structure,

$$A = \begin{pmatrix} A_{11} & & A_{13} \\ & A_{22} & A_{23} \\ A_{31} & A_{32} & A_{33} \end{pmatrix}.$$

The algorithm then continues recursively on the two subgraphs $\Omega_{1,1}$ and $\Omega_{1,2}$. We denote the separators subgraphs and the subdomains subgraphs introduced at level i of the nested dissection by $\Sigma_{i,j}$ and $\Omega_{i,l}$ respectively where $j \leq 2^{i-1}$, $l \leq 2^i$,$i \leq t$ and $t = log(P)$ (P number of processors). We also denote the vertices of the separators and the final subdomains by $\mathbf{S_{i,j}} = \mathbf{V}(\mathbf{\Sigma_{i,j}})$ and $\mathbf{D_l} = \mathbf{V}(\mathbf{\Omega_{t,l}})$ respectively. Thus at level i we introduce 2^{i-1} new separators and 2^i new subdomains. By disregarding the colored lines, Figure 1b presents the subdomains and the separators obtained by using three levels of nested dissection.

Although nested dissection splits the domain into independent subdomains that interact only with the separators, it is not sufficient to obtain a communication avoiding preconditioner. This can be seen in figure 1b. To compute one matrix-vector multiplication of the form $y_i = (LU)^{-1}Ay_{i-1}$ processor 1 has to fetch half of matrix A, half of matrix L, half of the vector y_{i-1} and almost the quarter of matrix U, and perform the associated computation. This shows that nested dissection alone is not sufficient to reduce communication for our matrix power kernel. This is because both the matrix vector multiplication and the forward/backward substitutions need to be performed in a communication avoiding manner. Note that in Metis library, the subdomains do not have a natural ordering as shown in Figure 1b, but they are partitioned again using nested dissection and this reduces the redundant flops (Appendix D). In what follows, we introduce our special reordering that reduces the ghost zones in figure 1b not only for performing 1 step, but also for perfoming $2, 3, .., s$ steps of the ILU matrix power kernel. This will allow the processors to perform s steps of the ILU0 matrix power kernel while avoiding any communication. We focus here on regular grids and we consider that $t = log(P)$ levels of nested dissection have been previously applied. Our reordering consists of two algorithms. The first rearranges the vertices of subdomains D_l

($l = 1, 2, ..., p$) obtained in the last level (t^{th} level) of Nested Dissection (Algorithm 3). The second rearranges the vertices of the separators $S_{j,m}$ where j is the level of nested dissection and m is the separator's order within this level of nested dissection ($m = 1, 2, ..., 2^{j-1}$) (Algorithm 4). In this manner, we can even perform the rearrangement in parallel where each processor rearranges its subdomain and then each can rearrange a separator. Note that we do not change neither the order of the domains and separators nor the set of indices assigned to each (refer to Appendix D). For example, in nested dissection, the indices 1 till 50 are assigned to the first domain (Figure 1b). In CA-ILU0(s) rearrangement, the indices 1 till 50 are still assigned to the first domain, however their ordering is changed (Figures 2a and 2b).

In a classic computation based on nested dissection, the computation on the subdomains is done in parallel, followed by the computation associated with the separators. This requires $\log(P)$ messages to be exchanged during the forward and the backward solves performed at each iteration of a Krylov subspace method. To be able to avoid communication, we first merge the computation of the separators to the subdomains. Therefore, each processor computes a set $\alpha_j = Adj(G(A), D_j) \cap (\cup_{\forall j} S_{j,m}) = Adj(G(A), D_j)$. Without going into details, the algorithm ensures that all the vertices of the separators belong to some α_j. For example in Figure 2a, nodes 231 and 473 are added to to some α_i.

The reordering is designed to isolate as much as possible the sets of vertices α_j, for all j, in the graphs of L and U. In other words, the goal is to minimize the number of vertices in the sets $\beta_j = R(G(U), \alpha_j)$ and $\gamma_j = R(G(L), \beta_j)$. For the U matrix, this means that the set β_j should be equal to the set $\alpha_j \cup h_{U,j,1}$, where $h_{U,j,1} = opAdj(G(U), \alpha_j)$. The data ghosted represents at most one layer of vertices around α_j. For this, the set $h_{U,j,1}$ is numbered with the largest possible numbers. By doing so, in 2D 5-point stencil and 9-point stencil grids, $h_{U,j,1}$ contains at most 4 vertices. For 3D 7-point stencil and 27-point stencil grids, $h_{U,j,1}$ is at most $12 \times \ (n/P)^{1/3} + 8$ vertices, where we assume in the first case that the subdomain is a cube containing n/P vertices and in the second case that α_j is a cube containing n/P vertices. Similarly, for the L matrix, the goal is to have the set γ_j to be as close as possible to the set $\beta_j \cup h_{L,j,1}$ (if possible equal), where $h_{L,j,1} = opAdj(G(L), \beta_j)$. Thus, the set $h_{L,j}$ is numbered with the smallest numbers possible. Hence one layer of ghosted data is added around β_j. By generating all these conditions for all α_j with $j = 1, 2, ..., p = 2^t$ and by taking into consideration the structure of a nested dissection graph, the reordering for the subdomains and the separators is presented in Algorithms 3 and 4 respectively, where the parameter s is set to 1. As it can be seen in Figure 2a, this alternating reordering reduces the ghost zones as compared to Figure 1b. Thus to compute one matrix-vector multiplication of the form $y_i = (LU)^{-1} A y_{i-1}$ on 8 processors, processor 1 has to fetch one eighth of matrix U, a bit more than one eighth of matrices L and A and of the vector y_{i-1}.

To perform $s - step$ of the multiplication $y_i = (LU)^{-1} A y_{i-1}$ in a communication-avoiding manner, our goal is to minimize the number of vertices in the sets $\beta_{j,i} = R(G(U), \alpha_{j,i})$ and $\gamma_{j,i} = R(G(L), \beta_{j,i})$, for $i = 1, 2, .., s$, $j = 1, .., 2^t$ where $\alpha_{j,1} = Adj(G(A), D_j)$ and for $i_1 > 1$, $\alpha_{j,i_1} = Adj(G(A), \gamma_{j,i_1-1})$. We perform the same analysis as for the case of $s = 1$, but for $s-steps$. The set $h_{U,j,i} = opAdj(G(A), \alpha_{j,i})$ is numbered with the largest possible numbers, and $h_{L,j,i} = opAdj(G(A), \beta_{j,i})$ is numbered with the smallest possible numbers, for $i = 1$ till s. This leads to an alternating rearrangement from the separators as shown in Figure 2b.

Algorithm 3 takes as input the graph of A, the vertices of the subdomain to be rearranged, the vertices of the separators. Note that to reorder a given subdomain D_j, the algorithm needs one separator from each level of nested dissection, specifically the separator which was part of a parent subdomain. The algorithm also takes as input s, the number of steps to be performed, and $evenodd$ which defines in which order we want to number our nodes "first, last, first,.. " (odd) or "last, first, last, first, ..." (even). Note that the first call to the algorithm to reorder a subdomain, the initial parameters are set to $evenodd = odd$ and num to be the set of indices assigned to the subdomain by dested dissection. Algorithm 3 is a recursive algorithm that starts by looping over the separators $S_{j,m}$ and finding their adjacent sets in D_l, bv_j. The aim is to number the bv_j's first (smallest indices) or last (largest indices) depending on the $evenodd$ tag which specifies if we are reducing $\beta_{j,k}$ or $\gamma_{j,k}$ ($k = 1, .., s$). In case some other separator $S_{i,m}$ depends on some

Algorithm 3 CA-ILU0subdomain (D_l, $S_{j,m}$ $\forall j, s, evenodd, num$)

Input: D_l, the set of vertices to be rearranged; G(A), the graph of A
Input: $S_{j,m}$ $\forall(j,m)$, the vertices of separators
Input: s, the number of multiplications to be performed without communication
Input: $evenodd$, a tag that can be either even or odd; num, the set of numbers/indices assigned to the vertices D_l

if $s == 0$ **then**
  Number D_l in any order, preferably in the natural order.
else
  for $j = 1$ to t **do** Find the vertices $bv_j = D_l \cap Adj_A(S_{j,m})$
  for $j = 1$ to t and $i = 1$ to t, $i \neq j$ **do** Find the vertices $cor_{j,i}$ = $bv_j \cap bv_i$, if they exist.
  for $j = 1$ to t , **do** let $corners_j = \cup_{\forall i}\ cor_{j,i}$
  for $j = 1$ to t **do**
    if evenodd = odd **then** Assign to the unnumbered vertices of bv_j, the smallest numbers in num, num_{bv_j}
    else Assign to the unnumbered vertices of bv_j, the largest numbers in num, num_{bv_j}
    end if
    Remove the numbers num_{bv_j} from num ($num = num - num_{bv_j}$)
    if $corners_j = \phi$ **then** Number the unnumbered vertices of bv_j with the indices num_{bv_j}, in any order.
    else Call **GeneralCA-ILU0Block** ($\boldsymbol{bv_j}$, $\boldsymbol{Sep_{i,m}}$ $\boldsymbol{\forall i \neq j, s, evenodd, num_{bv_j}}$)
    end if
  end for
  Let $D_l = D_l - \cup_{\forall j} bv_j$
  if evenodd = even **then** Call **GeneralCA-ILU0subdomain** ($\boldsymbol{D_l}$, $\boldsymbol{bv_j}$ $\boldsymbol{\forall j, s, odd, num}$)
  else Call **GeneralCA-ILU0subdomain** ($\boldsymbol{D_l}$, $\boldsymbol{bv_j}$ $\boldsymbol{\forall j, s-1, even, num}$)
  end if
end if

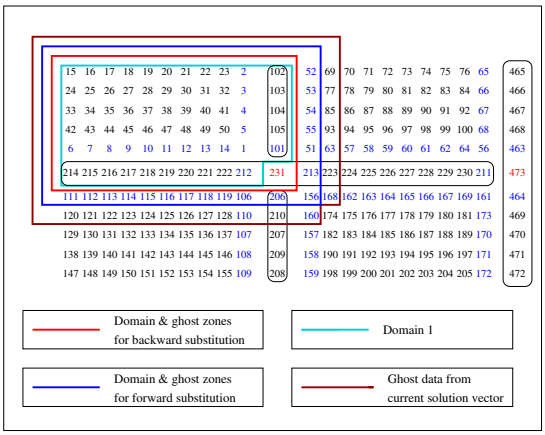
Algorithm 4 CA-ILU0Separator (S_{i,m_0}, $S_{j,m}$ $\forall(j,m)$,s,num)

Find all the interacting separators of $S_{i,m}$, iS_j where $j = 1, 2, ..., s$.
iS_j is the set of all boundary separators of D_o, $bS_{o,j} = Adj(G(A), D_o) \cap S_{j,m}$, where there is at least one vertex, $vert \in bS_{o,j}$, such that $vert \in\ Adj_A(S_{i,m})$.

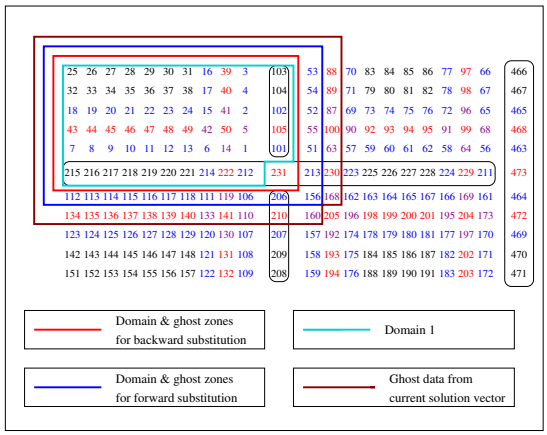
$$iS_j = \{bS_{o,j}\ \forall o, s.t. \exists vert \in bS_{o,j}\ and\ vert \in Adj_A(S_i(m))\}$$

for $j = 1$ to s, Find the set of vertices $int(i,m,j) = \begin{cases} S_i(m) \cap Adj_A(iS_j) & if\ iS_j \not\subseteq S_i(m) \\ S_i(m) \cap opAdj_A(iS_i) & if\ iS_j \subseteq S_i(m) \end{cases}$

Number the set $first = \{int(i,m,j); \forall\ j < i\}$ with the smallest numbers in num, num_1 and let $num = num - num_1$
for $j = i, i+1, ..., s$
  If for some $k < i$, $comm = int(i,m,j) \cap int(i,m,k) \neq \phi$ **then** Let $last1 = last1 \cup \{opAdj_A(comm), \forall comm \neq \phi\}$
  end if
  Number $last1$ with the largest numbers in num, num_{last1} and let $num = num - num_{last1}$
end for
Find the set $last2 = \{v \in int(i,m,j), \forall j > i\ \ \&\ \ v \notin int(i,m,j), \forall j \leq i\}$ and number it with the largest numbers in num, num_{last2}
Number the set of vertices $near = S_i(m) \cap opAdj_A(\ last1 \cup last2\ )$ with the smallest numbers in num, num_2 and let $num = num - num_2 - num_{last2}$
Let $bSep = \{near\ \cup\ last2\ \cup\ last1\ \cup\ first\}$
Let $Block = S_i(m) - bSep$
Call **GeneralCA-ILU0subdomain** ($\boldsymbol{Block}$, $\boldsymbol{bSep}$, $\boldsymbol{s-1, odd, num}$)



(a) The graph of the CA-ILU0(1) rearranged matrix A with the data needed to compute one matrix-vector multiplication on domain1



(b) The graph of the CA-ILU0(2) rearranged matrix A with the data needed to compute one matrix-vector multiplication on domain1

Figure 2: Half of an 11-by-43 5-point stencil grid, partitioned into 8 subdomains using 7 separators.

vertices of bv_j ($corners_j \neq \phi$) then we treat bv_j as a block, its separators being $S_{i,m}$, where $i \neq j$. Then Algorithm 3 is called recursively to limit the size of $\beta_{i,k}$ or $\gamma_{i,k}$. In case there is no separator that depends on bv_j, then we number it in any order. Finally, Algorithm 3 is called recursively on the remaining part of the subdomain $D_l - \cup_{\forall j} bv_j$ with the separators being bv_j, the appropriate value of $evenodd$ and s.

Algorithm 4 takes as input, the graph of A, the vertices of the separator to be rearranged S_{i,m_0}, the vertices of other separators $S_{j,m}$, and s. The aim is to find the vertices of S_{i,m_0} that belong to $h_{U,o,1}$ and $h_{L,o,1}$ for all o. Then the algorithm numbers $h_{U,o,1}$ last, $opAdj(G(A), h_{U,o,1})$ first, and $h_{L,o,1}$ first. This is done by looping over the separators from the same level j, iS_j that interact with S_{i,m_0} rather than the subdomains α_o. And we find $int(i,m,j)$, where $\cup_{\forall j} int(i,m,j) = \cup_{\forall o}(h_{U,o,1} \cup h_{L,o,1})$. After finding the vertices $int(i,m,j)$ and numbering them accordingly with $opAdj(G(A), last1 \cup last2)$ numbered last. In this way the vertices have been numbered for performing 1 step with no communication. Then Algorithm 3 is called to rearrange the remaning vertices of S_{i,m_0} alternatively.

A detailed analysis of the complexity of the CA-ILU0(s) reordering is given in appendix 7, where the complexity is defined as being the number of times the nodes and the edges in the graph of A are visited/read in order to perform the reordering. Taking into account that CA-ILU0(s) reordering is done in parallel on P processors, the parallel complexity is upper bounded by $2|D_t(l_{max})| + (3log(P) - 2)|S_{max}(m_{max})|$ where $D_t(l_{max})$ is the largest subdomain and $S_{max}(m_{max})$ is the largest separator. Note that for regular grids we may assume the all the the subdomains are of the same size. Thus our algorithm is of linear complexity.

4 The Expected Performance of the CA-ILU0 Preconditioner

The performance of the CA-ILU0 preconditioner depends on the convergence of GMRES for the CA-ILU0 preconditioned system, on the complexity of the CA-ILU0(s) rearrangement of the matrix A, and on the additional memory requirements and redundant flops of the ILU0 matrix power kernel.

Convergence It is known that the convergence of ILU0 depends on the ordering of the input matrix. The best convergence it is often observed when the matrix is ordered in natural ordering, while the usage of nested dissection tends to lead to a slower convergence (see for example [5]). We hence first discuss the effect of our reordering on the convergence of ILU0 preconditioned system. An s-step GMRES method can lead by itself to a slower convergence with respect to a classic GMRES method. Since our goal is to study the convergence of ILU0 preconditioner, we use in our experiments the classic GMRES method. Table 1 shows the real error, the relative residual, and the number of iterations of the preconditioned

Table 1: Convergence of the ILU0 preconditioned restarted GMRES on CA-ILU0(s) rearranged matrix"matvf2dNH100100". tol = 10^{-8}, maximum iterations = 200, number of restarts = 2

Ordering	**Real error** $\frac{\mathbf{norm(x_{sol}-x_{app})}}{\mathbf{norm(x_{sol})}}$	**Relative residual** $\frac{\mathbf{norm(b-Ax_{app})}}{\mathbf{norm(b)}}$	**Number of iterations**
NO	1.09×10^{-7}	9.80×10^{-9}	82
ND 16	8.19×10^{-7}	9.30×10^{-9}	148
ND 32	1.13×10^{-6}	8.80×10^{-9}	146
ND 64	1.45×10^{-6}	9.50×10^{-9}	142
CA-ILU0(1) 16	8.14×10^{-7}	9.50×10^{-9}	148
CA-ILU0(1) 32	1.31×10^{-6}	9.30×10^{-9}	147
CA-ILU0(1) 64	1.87×10^{-6}	9.70×10^{-9}	144
CA-ILU0(5) 16	1.43×10^{-6}	9.90×10^{-9}	147
CA-ILU0(5) 32	2.35×10^{-6}	9.10×10^{-9}	152
CA-ILU0(5) 64	2.47×10^{-6}	9.70×10^{-9}	149
CA-ILU0(10) 16	9.46×10^{-7}	9.40×10^{-9}	146
CA-ILU0(10) 32	2.44×10^{-6}	9.50×10^{-9}	152
CA-ILU0(10) 64	2.48×10^{-6}	9.70×10^{-9}	149

restarted GMRES on a 2D 5-point stencil matrix. This matrix, refered to as matvf2DNH100100, arises from a boundary value problem, and represents a non-homogeneous problem with large jumps in the coefficients, a more detailed description can be found in [1].

The first line shows the convergence of the ILU0 preconditioned GMRES where the matrix A is in its natural ordering (NO). The following three lines display the convergence of the ND-ILU0 preconditioned GMRES, where the matrix A is reordered using nested dissection (ND) with 3 different number of subdomains (16, 32 and 64). The remaining lines show the convergence of the CA-ILU0 preconditioned GMRES where the matrix A reordered using nested dissection and alternating reordering, with 3 different number of subdomains (16, 32 and 64). Three different sizes of s are used for CA-ILU0(s), $s = 1$, $s = 5$, and $s = 10$. We note that the real error and the relative residual of all the reordering strategies is of the same order. As for the number of iterations, as expected, the natural ordering ILU0 preconditioned system converges faster than ND-ILU0 and CA-ILU0(s). However ND-ILU0 and CA-ILU0(s) have similar rates of convergence. While we present here results for only one matrix due to space limits, we have performed experiments on a much larger set of matrices. We can conclude that our CA-ILU0 preconditioned system has a very similar convergence behavior as the ND-ILU0 system. It has the same rate of convergence, the same order of error and residual as the ND-ILU0 system. Thus, our extra rearrangement of the nested dissection matrix does not affect its convergence, while it enhances its communication avoiding parallelizability.

The memory requirements and redundant flops of the ILU0 matrix power kernel The ILU0 matrix power kernel avoids communication by performing redundant flops and storing more vectors and data.

Table 2 compares the needed memory and performed flops for "s" matrix vector multiplications on one subdomain/processor when using the non-preconditioned CA-GMRES and the CA-ILU0 preconditioned CA-GMRES on 2D 9-point stencils and 3D 27-point stencils. We assume that each processor has to compute the part $\alpha_{j,1} = Adj(G(A), D_j) \approx n/P = w^d$ of the"s" matrix vector multiplication, where $d = 2$ for 2D matrices and 3 for 3D matrices and w is the width of the square or cube. In CA-GMRES, we will store s vectors of the size $|R(G(A), \alpha, i-1)| \approx |(w + 2(i-1))^d|,\ i = 1, 2, ..., s$, one vector of the size $|\alpha|$ and the corresponding $|R(G(A), \alpha, s-1)|$ rows of the matrix A. Then, we will perform $\sum_{i=1}^{s}((w + 2(i-1))^d)(2 \times nnz - 1)$ flops where nnz is the number of nonzeros per row (9 and 27). For the CA-ILU0 preconditioned CA-GMRES, we store s vectors of size $|R(G(A), \alpha, 2(i-1))| \approx$

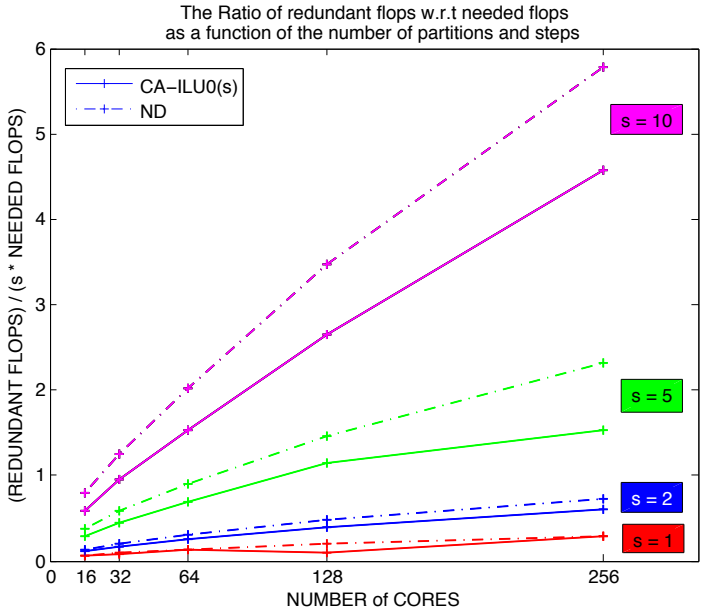


Figure 3: The ratio of redundant flops w.r.t needed flops in the ILU matrix power kernel for s= 1,2,5,10. The 2d matrix A = "matvf2dNH200200" is rearranged using Nested Disection and our CA-ILU0(s) rearrangement

$|w + 4(i-1))^d|$, $i = 1, 2, ..., s$, one vector of size α, 2 vectors of size $|R(G(A), \alpha, 2s-1)|$, the corresponding $|R(G(A), \alpha, 2s-1)|$ rows of the matrices A and L, and the $|R(G(A), \alpha, 2s-2)|$ rows of the matrix U. Then we will perform $\sum_{i=1}^{s}(2 \times nnz - 1)((w + 4i - 2)^d)$ flops to compute Ax. Solving the "s" lower triangular systems ($Lz = f$) requires $\sum_{i=1}^{s}[1 + 2 \times (nnz - 1)/2]((w + 4i - 2)^d)$ flops. Similarly, solving the "s" upper triangular systems requires $\sum_{i=1}^{s}[1 + 2 \times (nnz - 1)/2]((w + 4i - 4)^d)$ flops. Note that the memory and flops of CA-GMRES and CA-ILU0 CA-GMRES are governed by the same big O function.

Table 2: The memory and computational cost of performing "s" matrix vector multiplication on one subdomain, for the non-preconditioned CA-GMRES and of the CA-ILU0 preconditioned CA-GMRES

	CA-GMRES		**CA-ILU0 CA-GMRES**	
Stencil	**Memory**	**Flops**	**Memory**	**Flops**
2D 9-pt	$(s+10)\alpha + \frac{4}{3}s^3 + 38s^2 - \frac{214}{3}s + 36 + 2(s^2+19s-18)\alpha^{\frac{1}{2}}$	$17s\alpha + \frac{17}{3}s^3 + \frac{17}{6}(-3s^2+s) + 17(s^2-s)\alpha^{\frac{1}{2}}$	$(s+22)\alpha + \frac{16}{3}s^3 + 328s^2 + \frac{1256}{3}s + 144 + 2(2s^2+72s-52)\alpha^{\frac{1}{2}}$	$35s\alpha + \frac{560}{3}s^3 + 4s(35s-9)\alpha^{\frac{1}{2}} - 72s^2 - \frac{32}{3}s$
3D 27-pt	$(s+28)\alpha + 2s^4 + 2s(6s^2-11s+12) - 16 + 3(s^2+55s-54)\alpha^{\frac{2}{3}} + [4s^3 + 330s^2 - 650s + 324]\alpha^{\frac{1}{3}}$	$53s\alpha + 106s^4 + 106(-2s^3+s^2) + 159(s^2-s)\alpha^{\frac{2}{3}} + 106[2s^3-3s^2+s]\alpha^{\frac{1}{3}}$	$(s+58)\alpha + 16s^4 + 8s[452s^2 - 850s + 594] - 1240 + (6s^2+678s-426)\alpha^{\frac{2}{3}} + 4[4s^3 + 678s^2 - 850s + 297]\alpha^{\frac{1}{3}}$	$107s\alpha + 1712s^4 + 2s[321s-81]\alpha^{\frac{2}{3}} + [1712s^3 - 648s^2 - 968s]\alpha^{\frac{1}{3}} + 4720s - 1488s^2 - 2144s^3$

Figure 3 plots the ratio of the total redundant flops in the "ILU0 matrix power kernel" with respect to the needed flops in the "matrix vector multiplication of the sequential ILU0 preconditioned GMRES"

for s= 1,2,5,10 . The 2d 40000×40000 matrix A = "matvf2dNH200200" is rearranged using Nested Disection (ND) and our CA-ILU0(s) rearrangement. For all s, our CA-ILU0(s) rearrangement requires only (20 to 30%) less redundant flops than the Metis's ND rearrangement since Metis reduces redundant flops by performing much more than $log(P)$ levels of ND. However, our method is better since it also requires less memory per processor which leads to a reduction of the volume of the communicated data at the end of the s steps.Second as the s or the number of partitions increase, the redundant flops increase. Thus, one has to choose the appropriate number of partitions and steps s with respect to the problem at hand, to obtain the best performance. In other words, one has to find a balance between the redundant flops and communication.

5 Conclusion and Future Work

In this paper, we have introduced the first communication avoiding preconditioner, the CA-ILU0 preconditioner. First, we have adapted the matrix power kernel to the ILU preconditioned system to obtain the ILU matrix power kernel. Then we have introduced a rearrangement of the matrix A which is based on nested dissection. The CA-ILU0(s) rearrangement, rearranges the matrix A in a communication avoiding manner for performing s-steps of the multiplication $y_i = (LU)^{-1}Ay_{i-1}$. We have shown that rearrangement do not affect the convergence of the ILU0 preconditioned GMRES where the matrix A is rearranged using nested dissection. Then, we have shown that the complexity of the CA-ILU0(s) rearrangement is linear.We have also shown that CA-GMRES and CA-ILU0 preconditioned GMRES's memory requiremets and flops are limited by the same big O function. Not mentioning that performing s multiplications in CA-ILU0 preconditioned GMRES requires less total redundant flops than ND-ILU0 preconditioned GMRES. For all these reasons, we expect that our CA-ILU0 preconditioner will have a good performance. The kernels that we have introduced in this paper for the CA-ILU0 preconditioner, can be used to reduce or avoid communication for other solvers than CA-GMRES whenever the ILU0 matrix power kernel is used. Our future work will focus on implementing the CA-ILU0 preconditioner in a parallel environment to measure the improvements with respect to existing implementations. It also focuses on extending our reordering algorithm to unstructured grids. Then to extend the method to more general incomplete LU factorizations.

References

[1] Y. Achdou and F. Nataf. An iterated tangential filtering decomposition. Numerical Linear Algebra with Applications, 10(5-6):511539, 2003.

[2] E. Carson, N. Knight, and J. Demmel. Avoiding communication in two-sided krylov subspace methods. Technical Report UCB/EECS-2011-93, EECS Department, University of California, Berkeley, Aug 2011.

[3] J. Demmel, L. Grigori, M. Hoemmen, and J. Langou. Communication-avoiding parallel and sequential qr factorizations. SIAM J. Sci. Comput., 34 (2012), pp 206–239.

[4] J. Demmel, M. Hoemmen, M. Mohiyuddin, and K. Yelick. Avoiding communication in sparse matrix computations. In Parallel and Distributed Processing, 2008. IPDPS 2008. IEEE International Symposium on, pages 1 12, april 2008.

[5] S. Doi and T. Washio. Ordering strategies and related techniques to overcome the trade-off between parallelism and convergence in incomplete factorizations. Parallel Comput., 25(13-14):19952014, December 1999.

[6] J. Erhel. A parallel gmres version for general sparse matrices. Electronic Transactions on Numerical Analysis, 3:160176, 1995.

[7] L. Grigori, J. Demmel, and H. Xiang. CALU: A communication optimal LU factorization algorithm. SIAM J. Matrix Anal. Appl., 32(4):1317–13, November 2011.

[8] M. R. Hestenes and E. Stiefel. Methods of conjugate gradients for solving linear systems. Journal of research of the National Bureau of Standards, 49:409436, 1952.

[9] M. Hoemmen. Communication-Avoiding Krylov Subspace Methods. PhD thesis, EECS Department, University of California, Berkeley, 2010.

[10] M. S. Khaira, G. L. Miller, and T. J. Sheffler. Nested dissection: A survey and comparison of various nested dissection algorithms. Technical report, School of Computer Science, Carnegie Mellon University, 1992.

[11] M. Mohiyuddin, M. Hoemmen, J. Demmel, and K. Yelick. Minimizing communication in sparse matrix solvers. In Proceedings of the Conference on High Performance Computing Networking, Storage and Analysis, SC 09, pages 36:136:12, New York, NY, USA, 2009. ACM.

[12] Y. Saad. Iterative Methods for Sparse Linear Systems. Society for Industrial and Applied Mathematics, Philadelphia, PA, USA, 2nd edition, 2003.

[13] Y. Saad and M. H. Schultz. Gmres: a generalized minimal residual algorithm for solving nonsymmetric linear systems. SIAM J. Sci. Stat. Comput., 7(3):856869, July 1986.

[14] H. F. Walker. Implementation of the gmres method using householder transformations. SIAM J. Sci. Stat. Comput., 9(1):152163, January 1988.

6 Appendix A

Algorithm 5 is a simplified pseudocode of CA-GMRES. For the full version refer to [9, 11]. Thus the left preconditioned CA-GMRES pseudo code is that of CA-GMRES (Algorithm 5) with the replacement of $r_0 = b - Ax_0$ and $q_i = Aq_{i-1}$ by $r_0 = M^{-1}(b - Ax_0)$ and $q_i = M^{-1}Aq_{i-1}$

Algorithm 5 CA-GMRES

Input: $Ax = b$, $n \times n$ linear system
Input: x_0, the initial guess
Input: s, the size of the computed basis using the Matrix Power Kernel
Input: t, the number of outer iterations before restarting
Input: ϵ, convergence tolerence
Output: x_{si}, the approximate solution of the system $Ax = b$

Compute $r_0 = b - Ax_0$, $q_1 = r_0/||r_0||_2$, $\rho = ||r_0||_2$, $i = 0$
Perform an **Arnoldi(*s*,*t*)** iteration
  While ($\rho \geq \epsilon||b||_2$ **and** $i < t$)
    Compute $q_{si+2}, q_{si+3}, q_{si+4}, ..., q_{si+(s+1)}$ where $q_i = Aq_{i-1}$
      using **Matrix Power Kernel**
    Orthogonalize q_{si+j} $(2 \leq j \leq s+1)$ against q_j $(2 \leq j \leq si+1)$
      using **Block Gram Schmidt**
    Orthogonalize $q_{si+1}, q_{si+2}, ..., q_{si+(s+1)}$ using a **TSQR** factorization
    Reconstruct the upper Hessenberg matrix H_s
    Update ρ, $i = i + 1$
  end While
Solve the Least Square problem $y_{si} = min_y||P_s - H_s y||_2$
$x_{si} = x_0 + Q_s y_{si}$
  if ($\rho \geq \epsilon||b||_2$)
    Let $x_0 = x_{si}$ and restart by calling CA-GMRES (A, b, x_0, s, t, ϵ)
  else
    x_{si} is the approximate solution
  end if

7 Appendix B

The complexity of the CA-ILU0(s) rearrangement of the matrix A
We define the complexity of our CA-ILU0(s) rearrangement of the matrix A as being the number of times the nodes and the edges in the Graph of A are visited in order to perform the rearrangement. We would like to find this complexity in order to evaluate the ease at which we can perform our rearrangement.

We start by finding the complexity of rearranging $D_t(l)$ for $s - steps$. To define the alternating layers from the separators, we will have to read a maximum of $\sum_{\forall i} |R_A(bv_{i,l}, 2s-2)|$ nodes and their edges where $bv_{i,l} = Adj(G(A), S_i) \cap D_l$.

In case $corners_i \neq \phi$, we will have to read the $bv_{i,l}$ nodes and its edges again. So we will have to read some fraction of $|\cup_{\forall i} R_A(v(i,l), 2s-2)|$ nodes and its edges. In the worst case, we may assume that $corners(i) \neq \phi, \forall i$. Thus, we might read up to $\sum_{\forall i} |R_A(v(i,l), 2s-2)|$ nodes and their edges.

So in $D_t(l)$ we will read at most

$$2\sum_{\forall i} |R_A(v(i,l), 2s-2)| << 2|D_t(l)| << 2|D_t(l_{max})|$$

nodes and their edges, where $|D_t(l_{max})| > |D_t(l)|, \forall l \neq l_max$.

But before that, we need to read the separators $S_i(m_0), \forall i$ and find $bvi, l = Adj(G(A), S_i) \cap D_t$. In other words, we will visit the nodes of $S_i(m_0), \forall i$ and their edges. Each subdomain will read $log(p) = t$ separators where each separator is from a different level of nested dissection and $p = 2^t$ is the total number of subdomains.

So in total, S_1 will be read p times, $S_2(m)$ will be read $\frac{p}{2}$ times,..., $S_i(m)$ will be read $\frac{p}{2^{i-1}}$ times ($\sum_{\forall i=1}^{t} \frac{p}{2^{i-1}} \sum_{\forall m} |S_i(m)|$).

Let $S_{max}(m_{max})$ be a separator such that $|S_{max}(m_{max})| > |S_i(m)|,\ \forall (i, m)$ then

$$\sum_{\forall i=1}^{t} \frac{p}{2^{i-1}} \sum_{\forall m} |S_i(m)| < tp|S_{max}(m_{max})|$$

Thus to rearrange all the p subdomains we will need to visit/read the following nodes and their edges:

$$\begin{aligned} 2\sum_{\forall l=1}^{p} \sum_{\forall i} |R_A(v(i,l), 2s-2)| + \sum_{\forall i=1}^{t} \frac{p}{2^{i-1}} \sum_{\forall m} |S_i(m)| &<< 2\sum_{\forall l=1}^{p} |D_t(l)| + tp|S_{max}(m_{max})| \\ &<< 2p|D_{l_{max}}| + tp|S_{max}(m_{max})| \end{aligned}$$

As for the complexity of rearranging $S_i(m)$, we have to read the interacting separators $iSep(j), \forall j \neq i$ and their edges. Based on the nested dissection structure, $S_i(m)$ can interact at most with one separator from each level. Then, we have to read the separator itself to rearrange it for $s - steps$. So in total, to rearrange $S_i(m)$ we have to read $log(p) = t$ separators where each separator is from a different level of nested dissection and $p - 1 = 2^t - 1$ is the total number of separators.

Thus, for rearranging all the $p - 1$ separators, S_1 will be read $p - 1$ times, $S_2(m)$ will be read $\frac{p-2}{2} + 1$ times,..., $S_i(m)$ will be read $\frac{p-2^{i-1}}{2^{i-1}} + (i-1) = \frac{p}{2^{i-1}} + (i-2)$ times.

The total number of nodes to be read for rearranging all the separators can be expressed as:

$$\begin{aligned} \sum_{\forall i=1}^{t} (\frac{p}{2^{i-1}} + i - 2) \sum_{\forall m} |S_i(m)| &< \sum_{\forall i=1}^{t} (\frac{p}{2^{i-1}} + i - 2) 2^{i-1} |S_{max}(m_{max})| && (1) \\ &< |S_{max}(m_{max})|(tp + \sum_{\forall i=1}^{t} (i-2)2^{i-1}) && (2) \\ &< |S_{max}(m_{max})|(tp + 3 + (t-3)2^t) && (3) \\ &< |S_{max}(m_{max})|(tp + 3 + (t-3)p) && (4) \\ &< |S_{max}(m_{max})|(2tp + 3 - 3p)) && (5) \end{aligned}$$

For rearranging the matrix A we will have to read/visit the following nodes and their edges:

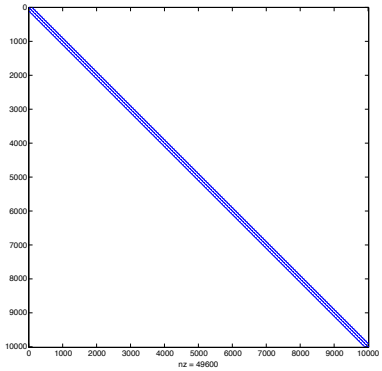
$$\begin{aligned} 2\sum_{\forall l=1}^{p} \sum_{\forall i} |R_A(v(i,l), 2s-2)| \quad + \quad & \sum_{\forall i=1}^{t} (\frac{2p}{2^{i-1}} + i - 2) \sum_{\forall m} |S_i(m)| \\ << \quad & 2p|D_t(l_{max})| + (3tp - 3p + 3)|S_{max}(m_{max})| \end{aligned}$$

Since the CA-ILU0(s) rearrangement is done in parallel on P processors, then this complexity will be divided by P.

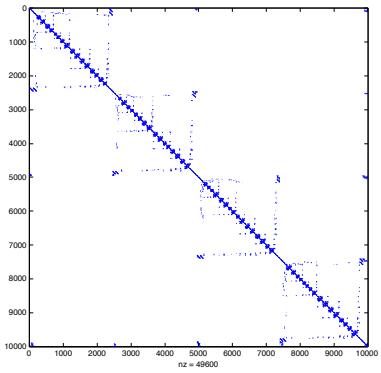
The parallel complexity of the CA-ILU0(s) rearrangement is less than

$$2|D_t(l_{max})| + (3t - 2)|S_{max}(m_{max})|$$

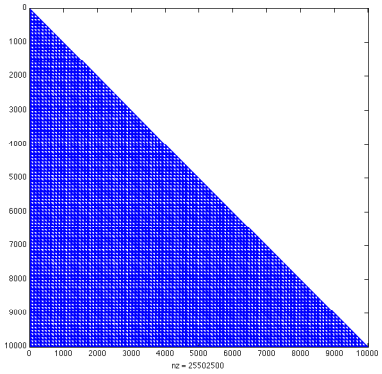
8 Appendix C



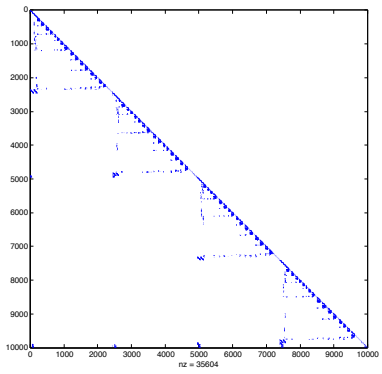
(a) 2D matrix A of size $10^4 \times 10^4$



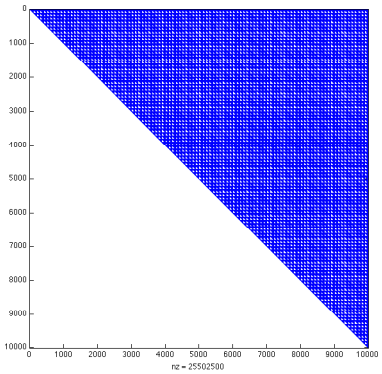
(b) A_{ca}, CA-ILU0(2) rearrangement of A



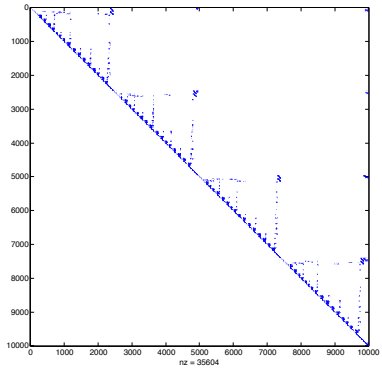
(c) L^{-1} matrix from the ILU(0) factorization of A



(d) L_{ca}^{-1} matrix from the ILU(0) factorization of A_{ca}



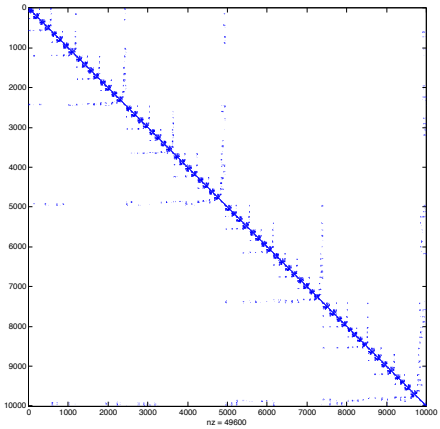
(e) U^{-1} matrix from the ILU(0) factorization of A



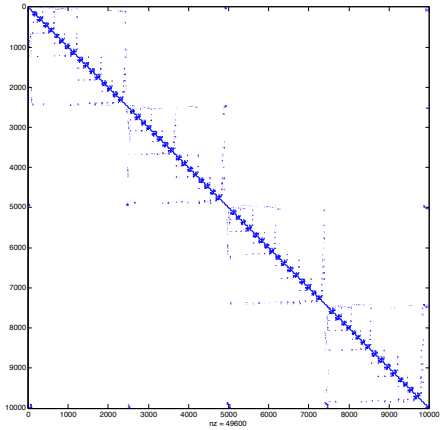
(f) U_{ca}^{-1} matrix from the ILU(0) factorization of A_{ca}

Figure 4: Comparison of the fill-ins in the ILU(0) factorization of a matrix A and its CA-ILU0(2) rearranged version A_{ca}

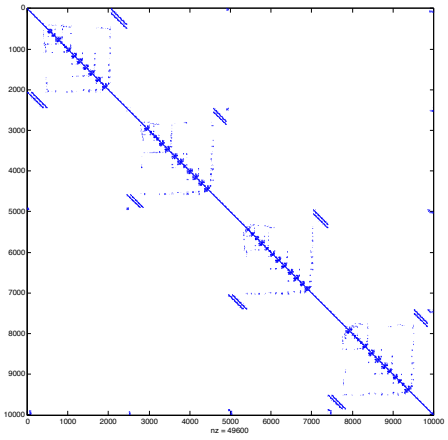
9 Appendix D



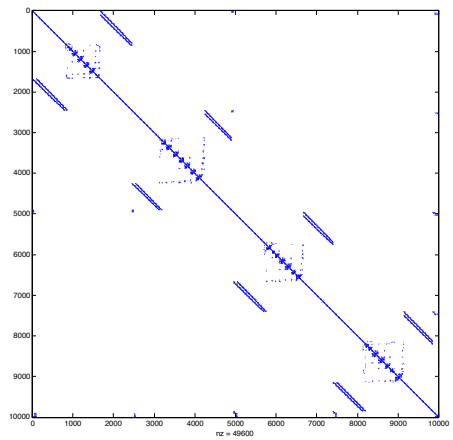
(a) Sparsity pattern of A_{nd}, Nested Dissection rearrangement of A with P=4



(b) Sparsity pattern of A_{ca1}, CA-ILU0(1) rearrangement of A with P=4

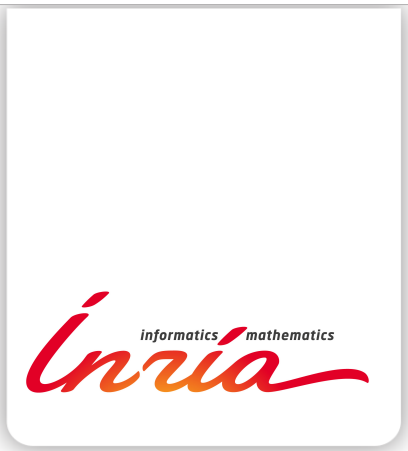


(c) Sparsity pattern of A_{ca5}, CA-ILU0(5) rearrangement of A with P=4



(d) Sparsity pattern of A_{c10}, CA-ILU0(10) rearrangement of A with P=4

Figure 5: Comparison of the sparsity patterns of the ND, CA-ILU0(1), CA-ILU0(5) and CA-ILU0(10) rearranged matrix matvf2DNH100100 of size $10^4 \times 10^4$



RESEARCH CENTRE
PARIS – ROCQUENCOURT

Domaine de Voluceau, - Rocquencourt
B.P. 105 - 78153 Le Chesnay Cedex

Publisher
Inria
Domaine de Voluceau - Rocquencourt
BP 105 - 78153 Le Chesnay Cedex
inria.fr

ISSN 0249-6399